Technical Document 2828
July 1995

Comparison of IEEE Portable Operating System Interface (POSIX) – Part I and X/OPEN Single UNIX Specifications (SUS)

D. K. Fisher
K. M. Tran

Prepared for:
Defense Information Systems Agency (DISA) Center for Standards

CONTENTS

1. INTRODUCTION

This document is a textual comparison study of the IEEP 1003.1 Portable Operating System Interface (POSIX) Part 1: System Application Program Interface [C Language] (IEEE 1003.1: 1990) and X/OPEN Single UNIX Specification (SUS). The purpose of this document is to aid in determining the criteria needed for the successor to FIPS PUB 151-2. It can also be used as a tool to ascertain the differences between the two specifications. However, note that this document does not attempt to address application portability concerns between the two specificiations. To determine the application portability of some commands between an XPG4 UNIX-Branded implementation and a FIPS 151-2 certified implementation,[1] further study is required.

1.1 SCOPE

This study consists of four components: (1) an introduction and outline to the document (2) a list of functions with no textual differences between IEEE 1003.1 and SUS, (3) an examination of functions with textual variances between IEEE 1003.1 and SUS, and (4) the appendices.

1.2 COMPARISON SUMMARY

Three parts (a Header, Description and Errors) make up each examination of a function's variances between IEEE 1003.1 and SUS. These sections are listed below, with an explanation of what is contained in each one. Note that the appearance of boldface text in any of the three sections signifies the addition of new partial text in SUS to existing IEEE 1003.1 text.

1.2.1 Header

This section notes the differences between *include* statements of IEEE 1003.1 and SUS. Three possible cases arise:

1. A IEEE 1003.1 *include* statement becomes an optional statement in SUS
2. A new *include* statement is added to SUS
3. There is no change.

A new function prototype is sometimes introduced in SUS with its own *include* statement, and will be so noted in this section.

1.2.2 Description

This section encompasses all the other differences not covered in the Header and Error sections, including conflicts of function descriptions, functionalities, and return values. Also covered are IEEE 1003.1 text omission and new SUS text additions.

1.2.3 Errors

When a function failure occurs, *errno* is set to the corresponding error code. This section details the new error codes and/or the new partial error description (in boldface) added to each function in SUS.

Two types of error codes are available. One is associated with fatal function failures; and the other, with possible function failures. The latter error code emphasizes that the function may or may not fail.

[1]In the event of any conflict in functionality, the *X/OPEN Single UNIX Specification* defers to the IEEE 1003.1 and ANSI C standards.

2. FUNCTIONS WITH NO MODIFICATIONS

No differences exist between IEEE 1003.1 and the SUS Single Unix Specification for the following functions:

abort()

cfgetispeed()

cfgetospeed()

ctermid()

dup()

dup2()

fpathconf()

ftell()

fwrite()

gets()

perror()

remove()

sigaddset()

sigdelset()

sigemptyset()

sigfillset()

sigismember()

siglongimp()

sigpending()

sigprocmask()

sigsetjmp()

sigsuspend()

time()

times()

uname()

3. FUNCTIONS WITH MODIFICATIONS

3.1 access()

3.1.1 Header

No change.

3.1.2 Description

No change.

3.1.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being executed.

3.2 alarm()

3.2.1 Header

#include <unistd.h> added.

3.2.2 Description

SUS text added:

Interactions between alarm() and any of setitimer(), ualarm() or usleep() are unspecified.

3.2.3 Errors

No change.

3.3 cfsetispeed(), cfsetospeed()

3.3.1 Header

No change.

3.3.2 Description

No change.

3.3.3 Errors

Additional error codes to function failure:

[EINVAL] The speed value is not a valid baud rate or outside the range of possible speed values as specified in <termios.h>.

3.4 chdir()

3.4.1 Header

#include <unistd.h> added.

3.4.2 Description

No change.

3.4.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[ENAMETOOLONG] Updated to also include the following: Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.5 chmod()

3.5.1 Header

#include <sys/types.h> made optional.

3.5.2 Description

SUS modifications:

Function also changes signal S_ISVTX in addition to S_ISUID, S_ISGID, and file permission bits. (IEEE 1003.1 Subclause 5.6.4.2, lines 800-801)

SUS text added:

If a directory is writable and the mode bit S_ISVTX is set on the directory, a process may remove or rename files within that directory only if one or more of the following is true:

The effective user ID of the process is the same as that of the owner ID of the file.

The effective user ID of the process is the same as that of the owner ID of the directory.

The process has appropriate privileges.

3.5.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[EINTR] A signal was caught during execution of the function.

[EINVAL] The value of the mode argument is invalid.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.6 chown()

3.6.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.6.2 Description

SUS text added:

> Changing the group ID is permitted to a process with an effective user ID equal to the user ID of the file, but without appropriate privileges, if and only if owner is equal to the file's user ID ***or (uid_t)-l*** and group is equal either to the calling process' effective group ID or to one of its supplementary group IDs.
>
> If owner or group is specified as (uid_t)-1 or (gid_t)-1 respectively, the corresponding ID of the file is unchanged.

IEEE 1003.1 and SUS differences:

> (IEEE 1003.1) Changing the owner [ID] is restricted to processes with appropriate privileges.
>
> (SUS) Changing the user ID is restricted to processes with appropriate privileges.
>
> (IEEE 1003.1) If -1 is return, no changes shall be made in the owner and group of the file.
>
> (SUS) If -1 is return, no changes are made in the user ID and group ID of the file.

3.6.3 Errors

Additional error codes for function failure:

> [ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

> [EIO] An I/O error occurred while reading or writing to the file system.
>
> [EINTR] The chown() function was interrupted by a signal which was caught.
>
> [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.7 close()

3.7.1 Header

#include <unistd.h> added.

3.7.2 Description

SUS text added:

> If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive a SIGPOLL signal for events associated with that STREAM, the calling process will be unregistered for events associated with the STREAM. The last close() for a STREAM causes the STREAM associated with *fildes* to be dismantled. If O_NONBLOCK is not set and there have been no signals posted for the STREAM, and if there is data on the module's write queue, close() waits for an unspecified time (for each module and driver) for any output to drain before dismantling the STREAM. The time delay can be changed via an I_SETCLTIME ioctl() request. If the O_NONBLOCK flag is set, or if there are any pending signals, close() does not wait for output to drain, and dismantles the STREAM immediately.

If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a pipe, the last close() causes a hangup to occur on the other end of the pipe. In addition, if the other end of the pipe has been named by fattach(), then the last close() forces the named end to be detached by fdetach(). If the named end has no open file descriptors associated with it and gets detached, the STREAM associated with that end is also dismantled.

If *fildes* refers to the master side of a pseudo-terminal, a SIGHUP signal is sent to the process group, if any, for which the slave side of the pseudo-terminal is the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal flushes all queued input and output.

If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message may be sent to the master.

3.7.3 Errors

Additional error codes to possible function failure:

[EIO] An I/O error occurred while reading from or writing to the file system.

3.8 closedir()

3.8.1 Header

#include <sys/types.h> made optional.

3.8.2 Description

No change.

3.8.3 Errors

Additional error codes to possible function failure:

[EINTR] The function was interrupted by a signal.

3.9 create()

3.9.1 Header

#include <sys/types.h> and #include <sys/stat.h> made optional.

3.9.2 Description

No change.

3.9.3 Errors

No change.

3.10 execl(), execle(), execlp(), execv(), execve(), execvp()

3.10.1 Header

#include <unistd.h> added.

3.10.2 Description

IEEE 1003.1 text (Subclause 3.1.2.2, lines 101-105) omitted in SUS:

> For the execle() function, the environment is provided by following the NULL pointer that shall terminate the list of arguments in the parameter list to execle() with an additional parameter, as if it were declared as:
>
> > char *const envp[]

SUS text added:

> If the process image file is not a valid executable object, execlp() and execvp() use the contents of that file as standard input to a command interpreter conforming to system(). In this case, the command interpreter becomes the new process image.
>
> The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:
>
> > setlocale(LC_ALL, “C”) is executed at startup.
>
> After a successful call to any of the exec functions, alternate signal stacks are not preserved and the SA_ONSTACK flag is cleared for all signals. (SUS appended this text to paragraph 124-128 of IEEE 1003.1 Subclause 3.1.2.2.)
>
> After a successful call to any of the exec functions, any functions previously registered by atexit() are no longer registered.
>
> If the ST_NOSUID bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID and saved set-group-ID are unchanged in the new process image. (SUS adds this text to beginning of paragraph 129-137 of IEEE 1003.1 Subclause 3.1.2.2.)
>
> Any shared memory segments attached to the calling process image will not be attached to the new process image.
>
> Any mappings established through mmap() are not preserved across an exec.
>
> The new process also inherits at least the following attributes from the calling process image:
>
> 1. nice value (see nice())
> 2. semadj values (see semop())
> 3. file size limit (see ulimit())
> 4. resource limits
> 5. controlling terminal
> 6. interval timers

3.10.3 Errors

Additional error codes to function failure:

> [ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

> [ENAMETOOLONG] The length of the path or file arguments, or an element of the environment variable PATH prefixed to a file, exceeds (PATH_MAX}, or a pathname component is longer than {NAME_MAX}.

[ETXTBSY] The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.

3.11 _exit(), exit()

3.11.1 Header

#include <unistd.h> added.

3.11.2 Description

SUS text added:

The exit() function first calls all functions registered by atexit(), in the reverse order of their registration. Each function is called as many times as it was registered.

If a function registered by a call to atexit() fails to return, the remaining registered functions are not called and the rest of the exit() processing is not completed. If exit() is called more than once, the effects are undefined.

The exit() function then flushes all output streams, closes all open streams, and removes all files created by tmpfile().

Each mapped memory object is unmapped.

Each attached shared-memory segment is detached and the value of shm_nattch (see shmget()) in the data structure associated with its shared memory ID is decremented by 1.

For each semaphore for which the calling process has set a semadj value, see semop(), that value is added to the semval of the specified semaphore.

If the parent process has set its SA_NOCLDWAIT flag, or set SIGCHLD to SIG_IGN the status will be discarded, and the lifetime of the calling process will end immediately.

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) All open file descriptors and directory streams in the calling process are closed.

(SUS) All of the file descriptors, directory streams, *conversion descriptors and message catalogue descriptors* open in the calling process are closed.

(IEEE 1003.1) If the parent process of the calling process is executing a wait() or waitpid(), it is notified of the termination of the calling process and the low order 8 bits of status are made available to it.

(SUS) If the parent process of the calling process is executing a wait(), *wait3(), waitid()* or waitpid(), *and has neither set its SA_NO CLD WA IT flag nor set SIGCHLD to SIG_IGN,* it is notified of the calling process' termination and the low-order eight bits (that is, bits 0377) of status are made available to it. If the parent is not waiting, the child's status will be made available to it when the parent subsequently executes wait(), *wait3(), waitid()* or waitpid().

(IEEE 1003.1) If the parent process of the calling process is not executing a wait() or waitpid()function, the exit status code is saved for return to the parent process whenever the parent process executes an appropriate subsequent wait() or waitpid().

(SUS) If the parent process of the calling process is not executing a wait(), *wait3(), waitid()* or waitpid(), and *has not set its SA_NO CLD WA IT flag, or set SIGCHLD to SIG_IGN,* the calling process is transformed into a zombie process. A zombie process is an inactive process and it will be deleted at some later time when its parent process executes wait(), *wait3(), waitid()* or waitpid().

(IEEE 1003.1) Children of a terminated process shall be assigned a new parent process ID, corresponding to an implementation-defined system process.

(SUS) The parent process ID of *all of the calling process' existing child processes and* zombie processes is set to the process ID of an implementation-dependent system process. That is, these processes are inherited by a special system process.

3.11.3 Errors

No change.

3.12 fclose()

Comparison is not possible because IEEE 1003.1 references this function to close(), whereas SUS details this function in its own section. However, SUS makes the following updates:

3.12.1 Errors

Modification of error codes to function failure:

[EFBIG] An attempt was made to write a file that exceeds the maximum file size or **the process' file size limit.**

Additional error codes to possible function failure:

[ENXIO] A request was made of a non-existent device, or the request was outside the device.

3.13 fcntl()

3.13.1 Header

#include <sys/types.h> made optional.

3.13.2 Description

No change.

3.13.3 Errors

No change.

3.14 fdopen()

3.14.1 Header

No change.

3.14.2 Description

IEEE 1003.1 text modified to SUS as follows:

r or **rb**	open a file for reading
w or **wb**	open a file for writing
aor **ab**	open a file for writing at end of file
or rb+ or r+b	open a file for update (reading and writing)
or wb+ or w+b	open a file for update (reading and writing)
or ab+ or a+b	open a file for update (reading and writing) at end of file

where the character b has no effect, but is allowed for ISO/IEC standard conformance.

3.14.3 Errors

Additional error codes to possible function failure:

[EBADF] *The fildes* argument is not a valid file descriptor.

[EINVAL] The mode argument is not a valid mode.

[EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

[EMFILE] {STREAM_MAX} streams are currently open in the calling process.

[ENOMEM] Insufficient space to allocate a buffer.

3.15 fflush()

3.15.1 Header

No change.

3.15.2 Description

No change.

3.15.3 Errors

Modification of error codes to function failure:

[EFBIG] An attempt was made to write a file that exceeds the maximum file size or **the process' file size limit.**

Additional error codes to possible function failure:

[ENXIO] A request was made of a non-existent device, or the request was outside the device.

3.16 fgetc(), fgets()

3.16.1 Header

No change.

3.16.2 Description

No change.

3.16.3 Errors

Additional error codes to function failure:

[EIO] This error code is updated in SUS to also include the occurrence of a physical I/O.

Additional error codes to possible function failure:

[ENOMEM] Insufficient storage space is available.

[ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

3.17 fileno()

3.17.1 Header

No change.

3.17.2 Description

No change.

3.17.3 Errors

Additional error codes to possible function failure:

[EBADF] The stream argument is not a valid stream.

3.18 fopen()

Comparison not possible because IEEE 1003.1 references this function to close(), whereas SUS details this function in its own section. However, SUS marks the following errors as additions:

3.18.1 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[EINVAL]	The value of the mode argument is not valid.
[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.
[EMFILE]	{STREAM_MAX} streams are currently open in the calling process.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.19 fork()

3.19.1 Header

#include <unistd.h> added.

3.19.2 Description

SUS text added:

> The child process may have its own copy of the parent's message catalogue descriptors.
>
> All *semadj* values are cleared.
>
> Interval timers are reset in the child process.

3.19.3 Errors

No change.

3.20 fprintf()

Comparison is not possible because IEEE 1003.1 references this function to write() and lseek(), whereas SUS details this function in its own section.

3.21 fputc()

Comparison is not possible because IEEE 1003.1 references this function to write() and lseek(), whereas SUS details this function in its own section. However, SUS marks the following errors as additions:

3.21.1 Errors

Additional error codes to function failure:

> [EFBIG] This error code is updated in SUS to also include the occurrence of writing to a file that exceeds the process' file size limit.
>
> [EIO] This error code is updated in SUS to also include the occurrence of a physical I/O.

Additional error codes to possible function failure:

> [ENOMEM] Insufficient storage space is available.
>
> [ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

3.22 fputs()

Comparison is not possible because IEEE 1003.1 references this function to write() and lseek(), whereas SUS details this function in its own section and with reference to fputc().

3.23 fread()

Comparison is not possible because IEEE 1003.1 references this function to read() and lseek(), whereas SUS details this function in its own section and with reference to fgetc().

3.24 freopen()

Comparison is not possible because IEEE 1003.1 references this function to fopen() and fclose(), whereas SUS details this function in its own section. However, SUS marks the following errors as additions:

3.24.1 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[EINVAL] The value of the mode argument is not valid.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.25 fscanf()

Comparison is not possible because IEEE 1003.1 references this function to lseek() and read(), whereas SUS details this function in its own section. However, SUS makes the following updates:

3.25.1 Description

Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence %n$, where n is a decimal integer in the range [1, {NL_ARGMAX}]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages. In format strings containing the %n$ form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.

The format can contain either form of a conversion specification, that is, % or %n$, but the two forms cannot normally be mixed within a single format string. The only exception to this is that %% or %* can be mixed with the %n$ form.

The following conversion characters are valid:

C Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The sequence is converted to a sequence of wide-character codes in the same manner as mbstowcs(). The corresponding argument must be a pointer to the initial wide-character code of an array of type wchar_t large enough to accept the sequence which is the result of the conversion. No null wide-character code is added. If the matched sequence begins with the initial shift state, the conversion is the same as expected for mbstowcs(); otherwise the behavior of the conversion is undefined. The normal skip over white-space characters is suppressed in this case.

S Matches a sequence of characters that are not white space. The sequence is converted to a sequence of wide character codes in the same manner as mbstowcs(). The corresponding argument must be a pointer to the initial wide-character code of an array of wchar_t large enough to accept the sequence and a terminating null widecharacter code, which will be added automatically. If the field width is specified, it denotes the maximum number of characters to accept.

3.25.2 Errors

Additional error codes to possible function failure:

[EILSEQ] Input byte sequence does not form a valid character.

[EINVAL] There are insufficient arguments.

3.26 fseek()

Comparison is not possible because IEEE 1003.1 references this function to lseek() and write(), whereas SUS details this function in its own section. However, SUS makes the following updates:

3.26.1 Description

If the stream is to be used with wide character input/output functions, offset must either be 0 or a value returned by an earlier call to ftell() on the same stream and whence must be SEEK_SET.

The fseek() function returns 0 if it succeeds; otherwise it returns **-1** and sets *errno* to indicate the error.

3.26.2 Errors

Additional error codes to function failure:

> [EFBIG] This error code is updated in SUS to also include the occurrence of writing to a file that exceeds the process' file size limit.
>
> [EIO] This error code is updated in SUS to also include the occurrence of a physical I/O.
>
> [ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

3.27 fstat()

3.27.1 Header

#include <sys/types.h> made optional.

3.27.2 Description

No change.

3.27.3 Errors

Additional error codes to function failure:

> [EIO] An I/O error occurred while reading from the file system.

Additional error codes to possible function failure:

> [EOVERFLOW] One of the values is too large to store into the structure pointed to by the buf argument.

3.28 getc()

Comparison is not possible because IEEE 1003.1 references this function to lseek() and read(), whereas SUS

references to fgetc().

3.29 getchar()

See getc().

3.30 getcwd()

3.30.1 Header

#include <unistd.h> added.

3.30.2 Description

No change.

3.30.3 Errors

Additional error codes to possible function failure:

[ENOMEM] Insufficient storage space is available.

3.31 getegid()

3.31.1 Header

#include <sys/types.h> made optional.

#include <unistd.h> added.

3.31.2 Description

No change.

3.31.3 Errors

No change.

3.32 getenv()

3.32.1 Header

No change.

3.32.2 Description

SUS adds the boldface text to the following:

> The getenv() function searches the environment list for a string of the form “name=value”, and returns a pointer to a string containing the value for the specified name. If the specified name cannot be found, a null pointer is returned. The string pointed to must not be modified by the application, but may be overwritten by a subsequent call to getenv() or **putenv()** but will not be overwritten by a call to any other function in this document.

3.32.3 Errors

No change.

3.33 geteuid()

3.33.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.33.2 Description

No change.

3.33.3 Errors

No change.

3.34 getgid()

3.34.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.34.2 Description

No change.

3.34.3 Errors

No change.

3.35 getgrgid(), getgrnam()

3.35.1 Header

#include <sys/types.h> made optional.

3.35.2 Description

SUS text added:

> On error, *errno* will be set to indicate the error.

3.35.3 Errors

Additional error codes to possible function failure:

> [EIO] An I/O error has occurred.
>
> [EINTR] A signal was caught during getgrgid().
>
> [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.
>
> [ENFILE] The maximum allowable number of files is currently open in the system.

3.36 getgroups()

3.36.1 Header

#include <sys/types.h> made optional.

#include <unistd.h> added.

3.36.2 Description

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) Upon successful completion, the number of supplementary group IDS is returned. This value is zero if {NGROUPS_MAX} is zero.

(SUS) A return value of 0 is no longer permitted, because {NGROUPS_MAX} cannot be 0.

3.36.3 Errors

No change.

3.37 getlogin()

3.37.1 Header

#include <unistd.h> added.

3.37.2 Description

SUS text added:

When the error occurs, SUS sets *errno* to indicate the event in addition to IEEE 1 003.1's returning of the NULL

pointer.

3.37.3 Errors

Additional error codes to possible function failure:

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENXIO] The calling process has no controlling terminal.

3.38 getpgrp(), getpid(), getppid()

3.38.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.38.2 Description

No change.

3.38.3 Errors

No change.

3.39 getpwnam(), getpwuid()

3.39.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.39.2 Description

SUS text added:

On error, *errno* is set to indicate the error.

3.39.3 Errors

Additional error codes to possible function failure:

[EIO] An I/O error has occurred.

[EINTR] A signal was caught during getpwnam().

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENFILE] The maximum allowable number of files is currently open in the system.

3.40 getquid()

3.40.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.40.2 Description

No change.

3.40.3 Errors

No change.

3.41 isatty()

3.41.1 Header

#include <sys/types.h> made optional.

3.41.2 Description

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) The isatty() function returns 1 *if fildes* is a valid file descriptor associated with a terminal; zero otherwise.

(SUS) The isatty() function returns 1 *if fildes* is associated with a terminal; otherwise it returns 0 and may set *errno* to indicate the error.

3.41.3 Errors

Additional error codes to possible function failure:

[EBADF] *The fildes* argument is not a valid open file descriptor.

[ENOTTY] *The fildes* argument is not associated with a terminal.

3.42 kill()

3.42.1 Header

#include <sys/types.h> made optional.

3.42.2 Description

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) If pid is -l, the behavior of the kill() function is unspecified.

(SUS) If pid is -1, sig will be sent to all processes (excluding an unspecified set of system processes) for which the process has permission to send that signal.

3.42.3 Errors

No change.

3.43 link()

3.43.1 Header

#include <unistd.h> added.

3.43.2 Description

No change.

3.43.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving pathl or path2.

[EXDEV] The link named by path2 and the file named by path 1 are on different file systems and the

implementation does not support links between file systems, or pathl **refers to a named STREAM.**

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.44 lseek()

3.44.1 Header

#include <sys/types.h> made optional.

3.44.2 Description

No change.

3.44.3 Errors

No change.

3.45 mkdir()

3.45.1 Header

#include <sys/types.h> made optional.

3.45.2 Description

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) The owner ID of the directory is set to the effective user ID of the process.

(SUS) The directory's user ID is set to the process' effective user ID.

3.45.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.46 mkfifo()

3.46.1 Header

#include <sys/types.h> made optional.

3.46.2 Description

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) The owner ID of the FIFO shall be set to the effective user ID of the process.

(SUS) The FIFO's user ID will be set to the process' effective user ID.

3.46.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.47 open()

3.47.1 Header

#include <sys/types.h> made optional.
#include <sys/stat.h> made optional.

3.47.2 Description

SUS text added:

If path refers to a STREAMS file, oflag may be constructed from O_NONBLOCK OR-ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS devices and have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.

If path names the master side of a pseudo-terminal device, then it is unspecified whether open() locks the slave side so that it cannot be opened. Portable applications must call unlockpt() before opening the slave side.

O_SYNC If O_SYNC is set on a regular file, writes to that file will cause the process to block until the data is delivered to the underlying hardware.

IEEE 1003.1 and SUS differences:

O_CREAT (IEEE 1003.l)

... The file permission bits shall be set to the value of mode except those set in the file mode creation mask of the process (see [umask()]). ...

O_CREAT (SUS)

... The access permission bits (see <sys/stat.h>) of the file mode are set to the value of the third argument taken as type mode_t modified as follows: a bitwise-AND is performed on the file-mode bits and the corresponding bits in the complement of the process' file mode creation mask. Thus, all bits in the file mode whose corresponding bit in the file mode creation mask is set are cleared. ...

3.47.3 Errors

Additional error codes to function failure:

[PIO] The path argument names a STREAMS file and a hangup or error occurred during the open().
[ELOOP] Too many symbolic links were encountered in resolving path.

[ENOSR] The path argument names a STREAMS-based file and the system is unable to allocate a STREAM.

[ENXIO] O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set and no process has the file open for reading. **The named file is a character special or block special file, and the device associated with this special file does not exist.**

Additional error codes to possible function failure:

[EAGAIN] The path argument names the slave side of a pseudo-terminal device that is locked. [EINVAL] The value of the oflag argument is not valid.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[ENOMEM] The path argument names a STREAMS file and the system is unable to allocate resources.

[ETXTBSY] The file is a pure procedure (shared text) file that is being executed and oflag is O_WRONLY or O_RDWR.

3.48 opendir()

3.48.1 Header

#include <sys/types.h> made optional.

3.48.2 Description

No change.

3.48.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.49 pathconf()

3.49.1 Header

No change.

3.49.2 Description

No change.

3.49.3 Errors

Additional error codes to function failure:

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.50 pause()

3.50.1 Header

#include <unistd.h> added.

3.50.2 Description

No change.

3.50.3 Errors

No change.

3.51 pipe()

3.51.1 Header

No change.

3.51.2 Description

SUS text added:

> It is unspecified whether fildes[0] is also open for writing and whether fildes[1] is also open for reading.

3.51.3 Errors

No change.

3.52 printf()

Comparison is not possible because IEEE 1003.1 references this function to lseek() and write(), whereas SUS references to fprintf().

3.53 putc()

Comparison is not possible because IEEE 1003.1 references this function to lseek() and write(), whereas SUS references to fputc().

3.54 putchar()

Comparison is not possible because IEEE 1003.1 references this function to lseek() and write(), whereas SUS references to putc().

3.55 puts()

Comparison is not possible because IEEE 1003.1 references this function to write() and lseek(), whereas SUS details this function in its own section and with reference to fputc().

3.56 read()

3.56.1 Header

#include <unistd.h> added to read().

SUS function and header added as follows:

```
#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

3.56.2 Description

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) If a read() is interrupted by a signal after it has successfully read some data, either it shall return - 1 with *errno* set to [EINTR], or it shall return the number of bytes read.

(SUS) If a read() is interrupted by a signal after it has successfully read some data, it will return the number of bytes read. (FIPS 151-2 conformance)

IEEE 1003.1 text omitted in SUS:

A read() from a pipe or FIFO shall never return with *errno* set to [EINTR] if it has transferred any data.

SUS text added to describe the new function and reading of data from STREAM files:

A read() from a STREAMS file can read data in three different modes: byte-stream mode, messagenondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the I_SRDOPT ioctl() request, and can be tested with the I_GRDOPT ioctl(). In byte-stream mode, read() retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, read() retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If read() does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next read() call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the read() returns is discarded, and is not available for a subsequent read(), readv() or getmsg() call.

How read() handles zero-byte STREAMS messages is determined by the current read mode setting. In bytestream mode, read() accepts data until it has read nbyte bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The read() function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next read(), readv() or getmsg(). In messagenondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A read() from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message. By default, STREAMs are in control-normal mode, in which a read() from a STREAMS file can only process messages that contain a data part but do not contain a control part. The read() fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the I_SRDOPT ioctl() command. In control-data mode, read() converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, read() discards message control parts but returns to the process any data part in the message.

In addition, read() and readv() will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of *errno* does not reflect the result of read() or readv() but reflects the prior error. If a hangup occurs on the STREAM being read, read() continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.

The readv() function is equivalent to read(), but places the input data into the iovcnt buffers specified by the members of the iov array: iov[0], iov[l],,,,,iov[iovcnt- l The iovcnt argument is valid if greater than 0 and less than or equal to { IOV_MAX }.

Each iovec entry specifies the base address and length of an area in memory where data should be placed. The readv() function always fills an area completely before proceeding to the next.

Upon successful completion, readv() marks for update the st_atime field of the file.

Upon successful completion, read() **and readv**() return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return - 1 and set *errno* to indicate the error. (Boldface text is added in SUS.)

3.56.3 Errors

Additional error codes to both functions' failure:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in read() or **readv().**

[EBADMSG] The file is a STREAM file that is set to control-normal mode and the message waiting to be read includes a control part.

[EINVAL] The STREAM or multiplexer referenced *by fildes* is linked (directly or indirectly) downstream from a multiplexer.

[EISDIR] The *fildes* argument refers to a directory and the implementation does not allow the directory to be read using read() or readv(). The readdir() function should be used instead.

Error codes to freadv() failure:

[EINVAL] The sum of the iov_len values in the iov array overflowed an ssize_t.

Error codes to both functions' possible failure:

[ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

Error codes to freadv()'s possible failure:

[EINVAL] The iovcnt argument was less than or equal to 0, or greater than { IOV_MAX }.

3.57 readdir()

3.57.1 Header

#include <sys/types.h> made optional.

3.57.2 Description

IEEE 1003.1 and SUS differences:

(IEEE 1003.1) The readdir() function shall not return directory entries containing empty names. It is unspecified whether entries are returned for dot or dot-dot.

(SUS) If entries for dot or dot-dot exist, one entry will be returned for dot and one entry will be returned for dotdot; otherwise they will not be returned.

(IEEE 1003.1) After a call to the fork() function, either the parent or the child (but not both) may continue processing the directory stream using readdir() or rewinddir() or both. If both the parent and child processes use these functions, the result is undefined. Either or both processes may use closedir()

(SUS) After a call to fork(), either the parent or child (but not both) may continue processing the directory stream using readdir(), rewinddir() or seekdir(). If both the parent and child processes use these functions, the result is undefined.

SUS text added:

If the entry names a symbolic link, the value of the d_ino member is unspecified.

3.57.3 Errors

Additional error codes to possible function failure:

[ENOENT] The current position of the directory stream is invalid.

3.58 rename()

3.58.1 Header

No change.

3.58.2 Description

SUS text added:

If old points to a pathname that names a symbolic link, the symbolic link is renamed. If new points to a pathname that names a symbolic link, the symbolic link is removed.

3.58.3 Errors

Additional error codes to function failure:

[EBUSY] The directory named by old or new is currently in use by the system or another process, and the implementation considers this an error, or **the file named by old or new is a named STREAM.**

[PIO] A physical I/O error has occurred.

[ELOOP] Too many symbolic links were encountered in resolving either pathname.

[EPERM] or [EACCES] The S_ISVTX flag is set on the directory containing the file referred to by old and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges; or new refers to an existing file, the S_ISVTX flag is set on the directory containing this file and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges.

Additional error codes to possible function failure:

> [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
>
> [ETXTBSY] The file to be renamed is a pure procedure (shared text) file that is being executed.

3.59 rewind()

Comparison is not possible because IEEE 1003. 1 references this function to lseek(), whereas SUS details this function in its own section and with reference to fseek().

3.60 rewinddir()

3.60.1 Header

#include <sys/types.h> made optional.

3.60.2 Description

IEEE 1003.1 and SUS differences:

> (IEEE 1003.1) After a call to the fork() function, either the parent or the child (but not both) may continue processing the directory stream using readdir() or rewinddir() or both. If both the parent and child processes use these functions, the result is undefined. **Either or both processes may use closedir().**
>
> (SUS) After a call to fork(), either the parent or child (but not both) may continue processing the directory stream using readdir(), rewinddir() or seekdir(). If both the parent and child processes use these functions, the result is undefined.

3.60.3 Errors

No change.

3.61 rmdir()

3.61.1 Header

#include <unistd.h> added.

3.61.2 Description

SUS text added:

> If path names a symbolic link, then rmdir() fails and sets *errno* to [ENOTDIR].

3.61.3 Errors

Additional error codes to function failure:

> [PIO] A physical I/O error has occurred.
>
> [ELOOP] Too many symbolic links were encountered in resolving path.

[EPERM] or [EACCES] The S_ISVTX flag is set on the parent directory of the directory to be removed and the caller is not the owner of the directory to be removed, nor is the caller the owner of the parent directory, nor does the caller have the appropriate privileges.

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.62 scanf()

Comparison is not possible because IEEE 1003.1 references this function to lseek() and read(), whereas SUS references this function to fscanf().

3.63 setgid()

3.63.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.63.2 Description

No change.

3.63.3 Errors

No change.

3.64 setlocale()

Comparison is not practical because SUS and IEEE 1003.1 descriptions for this function differ vastly. See actual documents.

3.65 setpgid()

3.65.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.65.2 Description

No change.

3.65.3 Errors

IEEE 1003.l error codes omitted in SUS:

[ENOSYS] The setpgid() function is not supported by this implementation.

3.66 setsid()

3.66.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.66.2 Description

No change.

3.66.3 Errors

No change.

3.67 **setuid**()

3.67.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.67.2 Description

No change.

3.67.3 Errors

No change.

3.68 **sigaction**()

3.68.1 Header

No change.

3.68.2 Description

SUS member function added to the minimum set contained in structure *sigaction:* void(*) (int, siginfo_t * void *) sa_sigaction (Signal-catching function)

Additional SUS sa_flags

SA_ONSTACK

If set and an alternate signal stack has been declared with sigaltstack() or sigstack(), the signal will be delivered to the calling process on that stack. Otherwise, the signal will be delivered on the current stack.

SA_RESETHAND

If set, the disposition of the signal will be reset to SIG_DFL and the SA_SIGINFO flag will be cleared on entry to the signal handler (Note: SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction). Otherwise, the disposition of the signal will not be modified on entry to the signal handler.

In addition, if this flag is set, sigaction() behaves as if the SA_NODEFER flag were also set.

SA_RESTART

This flag affects the behavior of interruptible functions; that is, those specified to fail with *errno* set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function will restart and will not fail with [EINTR] unless otherwise specified. If the flag is not set, interruptible functions interrupted by this signal will fail with *errno* set to [EINTR].

SA_SIGINFO

If cleared and the signal is caught, the signal-catching function will be entered as:

> void func(int signo); where signo is the only argument to the signal catching function. In this case the sa_handler member must be used to describe the signal catching function and the application must not modify the sa_sigaction member.

If SA_SIGINFO is set and the signal is caught, the signal-catching function will be entered as:

> void func(int signo, siginfo_t *info, void *context); where two additional arguments are passed to the signal catching function. If the second argument is not a null pointer, it will point to an object of type siginfo_t explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type ucontext_t to refer to the receiving process' context that was interrupted when the signal was delivered. In this case the sa_sigaction member must be used to describe the signal catching function and the application must not modify the sa_handler member.
>
> The si_signo member contains the system-generated signal number.
>
> The si_errno member may contain implementation-dependent additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
>
> The si_code member contains a code identifying the cause of the signal. If the value of si_code is less than or equal to 0, then the signal was generated by a process and si_pid and si_uid respectively indicate the process ID and the real user ID of the sender. The values of si_pid and si_uid are otherwise meaningless.
>
> SA_NOCLDWAIT
>
> If set, and sig equals SIGCHLD, child processes of the calling processes will not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited for children that were transformed into zombie processes, it will block until all of its children terminate, and wait(), wait3(), waitid() and waitpid() will fail and set *errno* to [ECHILD]. Otherwise, terminating child processes will be transformed into zombie processes, unless SIGCHLD is set to SIG_IGN.
>
> SA_NODEFER
>
> If set and sig is caught, sig will not be added to the process' signal mask on entry to the signal handler unless it is included in sa_mask. Otherwise, sig will always be added to the process' signal mask on entry to the signal handler.

SUS text added:

> When a signal is caught by a signal-catching function installed by sigaction(), a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either sigprocmask() or sigsuspend() is made). This mask is formed by taking the union of the current signal mask and the value of the sa_mask for the signal being delivered **unless SA_NODEFER or SA_RESETHAND is set,** and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.
>
> Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to sigaction()), **until the SA_RESETHAND flag causes resetting of the handler,** or until one of the exec functions is called.
>
> If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified, **except as specified below.**

If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes will not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited for children that were transformed into zombie processes, it will block until all of its children terminate, and wait(), wait3(), waitid() and waitpid() will fail and set *errno* to [ECHILD].

If SA_SIGINFO is cleared, the signal-catching function will be entered as:

void func(int signo); where func is the specified signal-catching function and signo is the signal number of the signal being delivered.

If SA_SIGINFO is set, the signal-catching function will be entered as:

void func(int signo, siginfo_t *siginfo, void *ucontextptr); where func is the specified signal-catching function, signo is the signal number of the signal being delivered, siginfo points to an object of type siginfo_t associated with the signal being delivered, and ucontextptr points to a ucontext_t.

The behavior of a process is undefined after it returns normally from a signal-catching function for a **SIGBUS,** SIGFPP, SIGILL or SIGSEGV signal that was not generated by kill() or raise().

Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving process will resume execution at the point it was interrupted unless the signal handler makes other arrangements. If longjmp() or _longjmp() is used to leave the signal handler, then the signal mask must be explicitly restored by the process.

POSIX.4- 1993 defines the third argument of a signal handling function when SA_SIGINFO is set as a void * instead of a ucontext_t *, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to uncontext_t *.

The BSD optional four argument signal handling function is not supported by this specification. The BSD declaration would be void handler(int sig, int code, struct sigcontext *scp, char *addr); where sig is the signal number, code is additional information on certain signals, scp is a pointer to the sigcontext structure, and addr is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when SA_SIGINFO is set.

3.68.3 Errors

No change.

3.69 sleep()

3.69.1 Header

#include <unistd.h> added.

3.69.2 Description

SUS text added:

Interactions between sleep() and any of setitimer(), ualarm() or usleep() are unspecified.

3.69.3 Errors

No change.

3.70 stat()

3.70.1 Header

#include <sys/types.h> made optional.

3.70.2 Description

No change.

3.70.3 Errors

Additional error codes to function failure:

[PlO] An error occurred while reading from the file system.

[ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[EOVERFLOW] A value to be stored would overflow one of the members of the stat structure.

3.71 sysconf()

3.71.1 Header

IEEE 1003.1 and SUS function prototype differences:

(IEEE 1003.1) long sysconf(int name);

(SUS) long int sysconf(int name);

3.71.2 Description

SUS adds the following system variables to the minimal set provided in IEEE 1003.1:

_POSIX2_C_BIND	_XOPEN_CRYPT	BC_SCALE_MAX
_POSIX2_C_DEV	_XOPEN_ENH_I1 8N	BC_STRING_MAX
_POSIX2_C_VERSION	_XOPEN_SHM	COLL_WEIGHTS_MAX
_POSIX2_CHAR_TERM	_XOPEN_UNIX	EXPR_NEST_MAX
_POSIX2_FORT_DEV	_XOPEN_VERSION	IOV_MAX
_POSIX2_FORT_RUN	_XOPEN_XCU_VERSION	LINE_MAX
_POSIX2_LOCALEDEF	ATEXIT_MAX	PAGESIZE
_POSIX2_SW_DEV	BC_BASE_MAX	POSIX2_UPE
_POSIX2_VERSION	BC_DIM_MAX	RE_DUP_MAX

SUS text added:

If the value of:

sysconf(_SC_2_VERSION) is not equal to the value of the {_POSIX2_VERSION} symbolic constant, the utilities available via system() or popen() might not behave as described in the XCU specification. This would mean that the application is not running in an environment that conforms to the XCU specification. Some applications might be able to deal with this, others might not. However, the interfaces defined in this document will continue to operate as specified, even if:

sysconf(_SC 2_VERSION) reports that the utilities no longer perform as specified.

IEEE 1003.1 and SUS differences:

(IEEE 1 003.1) The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's <limits.h> or unistd.h>.

(SUS) The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's <limits.h>, unistd.h>, or **<time.h>.**

3.71.3 Errors

No change.

3.72 tcdrain()

3.72.1 Header

No change.

3.72.2 Description

No change.

3.72.3 Errors

Additional error codes to possible function failure:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

3.73 tcflow()

3.73.1 Header

No change.

3.73.2 Description

SUS text added:

Attempts to use tcflow() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

3.73.3 Errors

Additional error codes to possible function failure:

> [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

3.74 tcflush()

3.74.1 Header

No change.

3.74.2 Description

SUS text added:

> Attempts to use tcflush() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

3.74.3 Errors

Additional error codes to possible function failure:

> [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

3.75 tcgetattr()

3.75.1 Header

No change.

3.75.2 Description

IEEE 1003.1 and SUS differences:

> (IEEE 1003.1) The rate returned as the input baud rate [if differing baud rates are not supported] shall be either the number zero or the output rate.

> (SUS) If the terminal device does not support split baud rates, the input baud rate stored in the termios structure will be 0.

3.75.3 Errors

Additional error codes to possible function failure:

> [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

3.76 tcgetpgrp()

3.76.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.76.2 Description

No change.

3.76.3 Errors

IEEE 1003.l error codes omitted in SUS:

> [ENOSYS] The tcgetpgrp() function is not supported in this implementation.

3.77 tcsendbreak()

3.77.1 Header

No change.

3.77.2 escription

SUS text added:

> Attempts to use tcsendbreak() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

3.77.3 Errors

Additional error codes to possible function failure:

> [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

3.78 tcsetattr()

3.78.1 Header

No change.

3.78.2 Description

SUS text added:

> Attempts to use tcsetattr() from a process which is a member of a background process group on *a fildes* associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

3.78.3 Errors

Additional error codes to possible function failure:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

3.79 tcsetpgrp()

3.79.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.79.2 Description

No change.

3.79.3 Errors

IEEE 1003.l error codes omitted in SUS:

[ENOSYS] The tcsetpgrp() function is not supported in this implementation.

3.80 tmpfile()

Comparison is not possible because IEEE 1003.1 references this function to fopen(), whereas SUS details this function in its own section. However, SUS makes the following updates:

3.80.1 Errors

Additional error codes to possible function failure:

[EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

3.81 ttyname()

3.81.1 Header

#include <sys/types.h> made optional.

3.81.2 Description

IEEE 1003.l and SUS differences:

(IEEE 1003.1) The ttyname() function returns a NULL pointer if *fildes* is not a valid file descriptor associated with a terminal or if the pathname cannot be determined.

(SUS) Upon successful completion, ttyname() returns a pointer to a string. Otherwise, a null pointer is returned and errno is set to indicate the error.

3.81.3 Errors

Additional error codes to possible function failure:

[EBADF] *The fildes* argument is not a valid open file descriptor.

[ENOTTY] *The fildes* argument is not associated with a terminal.

3.82 tzset()

3.82.1 Header

No change.

3.82.2 Description

SUS text added:

> The tzset() function also sets the external variable daylight to 0 if Daylight Savings Time conversions should never be applied for the time zone in use; otherwise non-zero. The external variable timezone is set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time, for example:

TZ	timezone
EST	5*60*60
GMT	0*60*60
JST	-9*60*60
MET	-1*60*60
MST	7*60*60
PST	8*60*60

3.82.3 Errors

No change.

3.83 umask()

3.83.1 Header

#include <sys/types.h> made optional.

3.83.2 Description

No change.

3.83.3 Errors

No change.

3.84 unlink()

3.84.1 Header

#include <unistd.h> added.

3.84.2 Description

SUS text added:

> If path names a symbolic link, unlink() removes the symbolic link named by path and does not affect any file or directory named by the contents of the symbolic link

3.84.3 Errors

Additional error codes to function failure:

> [EBUSY] The file named by the path argument cannot be unlinked because it is being used by the system or another process and the implementation considers this an error, or **the file named by path is a named STREAM.**
>
> [ELOOP] Too many symbolic links were encountered in resolving path.
>
> [EPERM] or [EACCES] The S_ISVTX flag is set on the directory containing the file referred to by the path argument and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges.

Additional error codes to possible function failure:

> [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
>
> [ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.

3.85 utime()

3.85.1 Header

#include <sys/types.h> made optional.
#include <unistd.h> added.

3.85.2 Description

IEEE 1003.1 text omitted:

> Implementations may add extensions as permitted (in Subclause 1 .3.1.1, point (2)). Adding extensions to this structure, which might change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, also requires that the extensions be enabled as required by the same subclause above.

3.85.3 Errors

Additional error codes to function failure:

> [ELOOP] Too many symbolic links were encountered in resolving path.

Additional error codes to possible function failure:

> [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

3.86 wait(), waitpid()

3.86.1 Header

#include <sys/types.h> made optional.

3.86.2 Description

SUS text added:

WCONTINUED The waitpid() function will report the status of any continued child process specified by pid whose status has not been reported since it continued from a job control stop.

WIFCONTINUED(stat_val) Evaluates to a non-zero value if status was returned for a child process that has continued from a job control stop.

If the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, and the process has no unwaited for children that were transformed into zombie processes, it will block until all of its children terminate, and wait() and waitpid() will fail and set *errno* to [ECHILD].

If the information pointed to by stat_loc was stored by a call to waitpid() that specified the WUNTRACED flag **and did not specify the WCONTINUED flag,** exactly one of the macros WIFEXITED(*stat_loc), WIFSIGNALED(*stat_loc), and WIFSTOPPED(*stat_loc), will evaluate to a non-zero value.

If the information pointed to by stat_loc was stored by a call to waitpid() that specified the WUNTRACED **and WCONTINUED** flags, exactly one of the macros WIFEXITED(*stat_loc), WIFSIGNALED(*stat_loc), WIFSTOPPED(*stat_loc), **and WIFCONTINUED(*stat_loc),** will evaluate to a non-zero value.

If the information pointed to by stat_loc was stored by a call to waitpid() that did not specify the WUNTRACED **or WCONTINUED flags,** or by a call to the wait() function, exactly one of the macros WIFEXITED(*stat_loc) and WIFSIGNALED(*stat_loc) will evaluate to a non-zero value.

If the information pointed to by stat_loc was stored by a call to waitpid() that did not specify the WUNTRACED flag **and specified the WCONTINUED flag,** or by a call to the wait() function, exactly one of the macros WIFEXITED(*stat_loc), WIFSIGNALED(*stat_loc), **and WIFCONTINUED(*stat_loc),** will evaluate to a non-zero value.

3.86.3 Errors

No change.

3.87 write()

3.87.1 Header

#include <unistd.h> added.
SUS function and header added as follows:

```
#include <sys/uio.h>

ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);
```

3.87.2 Description

IEEE 1 003.1 and SUS differences:

(IEEE 1003.1) If write() is interrupted by a signal after it successfully writes some data, either it shall return-1 with *errno* set to [EINTR], or it shall return the number of bytes written. A write() to a pipe or FIFO shall never return with *errno* set to [EINTR] if it has transferred any data and nbyte is less than or equal to {PIPE_BUF}.

(SUS) If write() is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

SUS text added:

If the O_SYNC flag of the file status flags is set *and fildes* refers to a regular file, a successful write() does not return until the data is delivered to the underlying hardware.

If a write() requests that more bytes be written than there is room for (for example, **the ulimit or** the physical end of a medium), only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes will give a failure return (except as noted below) **and the implementation will generate a SIGXFSZ signal for the process.**

If fildes refers to a STREAM, the operation of write() is determined by the values of the minimum and maximum nbyte range ("packet size") accepted by the STREAM. These values are determined by the topmost STREAM module. If nbyte falls within the packet size range, nbyte bytes will be written. If nbyte does not fall within the range and the minimum packet size value is 0, write() will break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). **If** nbyte does not fall within the range and the minimum value is non-zero, write() will fail with *errno* set to [ERANGE]. Writing a zero-length buffer (nbyte is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue I_SWROPT ioctl() to enable zero-length messages to be sent across the pipe or FIFO.

When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:

If O_NONBLOCK is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), write() will block until data can be accepted.

If O_NONBLOCK is set and the STREAM cannot accept data, write() will return - 1 and set *errno* to [EAGAIN].

If O_NONBLOCK is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, write() will terminate and return the number of bytes written.

In addition, write() and writev() will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of *errno* does not reflect the result of write() or writev() but reflects the prior error.

The writev() function is equivalent to write(), but gathers the output data from the iovcnt buffers specified by the members of the iov array: iov[0], iov[1],,,,iov[iovcnt-l]. iovcnt is valid if greater than 0 and less than or equal to {IOV_MAX}, defined in <limits.h>.

Each iovec entry specifies the base address and length of an area in memory from which data should be written. The writev() function will always write a complete area before proceeding to the next.

If fildes refers to a regular file and all of the iov_len members in the array pointed to by iov are 0, writev() will return 0 and have no other effect. For other file types, the behavior is unspecified.

If the sum of the iov_len values is greater than SSIZE_MAX, the operation fails and no data is transferred.

Upon successful completion, writev() returns the number of bytes actually written. Otherwise, it returns a value of - 1, the file-pointer remains unchanged, and *errno* is set to indicate an error.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, *errno* is set to the value included in the error message.

3.87.3 Errors

IEEE 1003.1 existing error codes for write() also applies to function writev().

Error codes to both functions' failure:

[EFBIG] An attempt was made to write a file that exceeds the implementation-dependent maximum file size or **the process' file size limit.**

[EIO] A physical I/O error has occurred.

[EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that **only has one end open.** A SIGPIPE signal will also be sent to the process.

[ERANGE] The transfer request size was outside the range supported by the STREAMS file associated with *fildes.*

Error codes to writev()'s failure:

[EINVAL] The sum of the iov_len values in the iov array would overflow an ssize_t.

Error codes to both functions' possible failure:

[EINVAL] The STREAM or multiplexer referenced *by fildes* is linked (directly or indirectly) downstream from a multiplexer.

[ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

[ENXIO] A hangup occurred on the STREAM being written to.

Error codes to writev()'s possible failure:

[EINVAL] The iovcnt argument was less than or equal to 0, or greater than {IOV_MAX}.

4. CROSS REFERENCE TABLE

The following table summarizes all the functions' examinations in a table for cross-referencing purposes.

Table 4-1. Functionality change details.

NUM	FUNCTIONS	NO CHANGE	SECTIONS CHANGED HEADER	SECTIONS CHANGED DESC	SECTIONS CHANGED ERROR	COMMENTS	REF PAGE
1	_exit( )		X	X			7
2	abort( )	X					2
3	access( )				X		3
4	alarm( )		X	X			3
5	cfgetispeed( )	X					2
6	cfgetospeed( )	X					2
7	cfsetispeed( )				X		3
8	cfsetospeed( )				X		3
9	chdir( )		X		X		4
10	chmod( )		X	X	X		4
11	chown( )		X	X	X		5
12	close( )		X	X	X		5
13	closedir( )		X		X		6
14	creat( )		X				6
15	ctermid( )	X					2
16	dup( )	X					2
17	dup2( )	X					2
18	execl( )		X	X	X		6
19	execle( )		X	X	X		6
20	execlp( )		X	X	X		6
21	execv( )		X	X	X		6
22	execve( )		X	X	X		6
23	execvp( )		X	X	X		6
24	exit( )		X	X			7
25	fclose( )				X	Comparison N/A	9
26	fcntl( )		X				9
27	fdopen( )			X	X		9
28	fflush( )				X		10
29	fgetc( )				X		10
30	fgets( )				X		10
31	fileno( )				X		10
32	fopen( )				X	Comparison N/A	11

Table 4-1. Functionality change details (Continued).

33	fork( )		X	X			11
34	fpathconf( )	X					2
35	fprintf( )					Comparison N/A	11
36	fputc( )				X	Comparison N/A	11
37	fputs( )					Comparison N/A	12
38	fread( )					Comparison N/A	12
39	freopen( )				X	Comparison N/A	12
40	fscanf( )			X	X	Comparison N/A	12
41	fseek( )			X	X	Comparison N/A	13
42	fstat( )		X		X		13
43	ftell( )	X					2
44	fwrite( )	X					2
45	getc( )					Comparison N/A	14
46	getchar( )					Comparison N/A	14
47	getcwd( )		X		X		14
48	getegid( )		X				14
49	getenv( )			X			14
50	geteuid( )		X				15
51	getgid( )		X				15
52	getgrgid( )		X	X	X		15
53	getgrnam( )		X	X	X		15
54	getgroups( )		X	X			16
55	getlogin( )		X	X	X		16
56	getpgrp( )		X				16
57	getpid( )		X				16
58	getppid( )		X				16
59	getpwnam( )		X	X	X		17
60	getpwuid( )		X	X	X		17
61	gets( )	X					2
62	getquid( )		X				17
63	isatty( )		X	X	X		17
64	kill( )		X	X			18
65	link( )		X		X		18
66	lseek( )		X				18
67	mkdir( )		X	X	X		18
68	mkfifo( )		X	X	X		19
69	open( )		X	X	X		19
70	opendir( )		X		X		20
71	pathconf( )				X		21
72	pause( )		X				21

Table 4-1. Functionality change details (Continued).

73	perror( )	X					2
74	pipe( )			X			21
75	printf( )					Comparison N/A	22
76	putc( )					Comparison N/A	22
77	putchar( )					Comparison N/A	22
78	puts( )					Comparison N/A	22
79	read( )		X	X	X		22
80	readdir( )		X	X	X		24
81	remove( )	X					2
82	rename( )			X	X		24
83	rewind( )					Comparison N/A	25
84	rewinddir( )		X	X			25
85	rmdir( )		X	X	X		25
86	scanf( )					Comparison N/A	26
87	setgid( )		X				26
88	setlocale( )					Comparison N/A	26
89	setpgid( )		X		X		26
90	setsid( )		X				26
91	setuid( )		X				27
92	sigaction( )			X			27
93	sigaddset( )	X					2
94	sigdelset( )	X					2
95	sigemptyset( )	X					2
96	sigfillset( )	X					2
97	sigismember( )	X					2
98	siglongjmp( )	X					2
99	sigpending( )	X					2
100	sigprocmask( )	X					2
101	sigsetjmp( )	X					2
102	sigsuspend( )	X					2
103	sleep( )		X	X			29
104	stat( )		X		X		29
105	sysconf( )		X	X			30
106	tcdrain( )				X		31
107	tcflow( )			X	X		31
108	tcflush( )			X	X		31
109	tcgetattr( )			X	X		32
110	tcgetpgrp( )		X		X		32
111	tcsendbreak( )			X	X		32
112	tcsetattr( )			X	X		33

Table 4-1. Functionality change details (Continued).

113	tcsetpgrp( )		X		X		33
114	time( )	X					2
115	times( )	X					2
116	tmpfile( )				X	Comparison N/A	33
117	ttyname( )		X	X	X		33
118	tzset( )			X			34
119	umask( )		X				34
120	uname( )	X					2
121	unlink( )		X	X	X		34
122	utime( )		X	X	X		35
123	wait( )		X	X			35
124	waitpid( )		X	X			35
125	write( )		X	X	X		36

5. REFERENCES

{1} IEEE 1003.1 Portable Operating System Interface - Part I: System Application Program Interface [C Language]

{2} SUS Single UNIX Specification (on CD-ROM)

{3} Federal Information Processing Standards Publication 151-2

6. BIBLIOGRAPHY

{1} Go Solo-How to Implement & Go Solo with the Single UNIX Specification, by Stephen R. Walli

REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE July 1995	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE COMPARISON OF IEEE PORTABLE OPERATING SYSTEM INTERFACE (POSIX) – PART I and X/OPEN SINGLE UNIX SPECIFICATIONS (SUS)		5. FUNDING NUMBERS PE: 0604574N AN: DN302171
6. AUTHOR(S) D. K. Fisher, K. M. Tran		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Command, Control and Ocean Surveillance Center (NCCOSC) RDT&E Division San Diego, CA 92152–5001		8. PERFORMING ORGANIZATION REPORT NUMBER TD 2828
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Information Systems Agency (DISA) Center for Standards Parkridge Boulevard Reston, VA 22091–4398		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document is a textual comparison study of the IEEE 1003.1 Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface [C Language] (IEEE 1003.1: 1990) and X/OPEN Single UNIX Specification (SUS). The purpose of this document is to aid in determining the criteria needed for the successor to FIPS PUB 151-2. It can also be used as a tool to ascertain the differences between the two specifications. However, note that this document does not attempt to address application portability concerns between the two specifications. To determine the application portability of some commands between an XPG4 UNIX-Branded implementation and a FIPS 151-2 certified implementation, further study is required.

14. SUBJECT TERMS Portable Operating System Interface (POSIX) – Part I X/OPEN Single UNIX Specification (SUS) FIPS 151–2 Specification			15. NUMBER OF PAGES 52 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAME AS REPORT